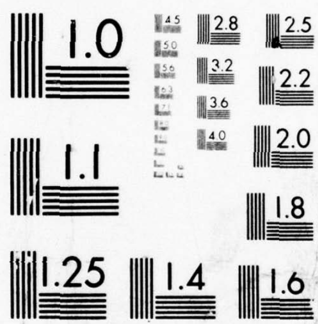
AD-A069 861   YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE   F/G 5/7
THE ROLE OF OBJECT PRIMITIVES IN NATURAL LANGUAGE PROCESSING.(U)
MAY 79   W G LEHNERT, M H BURSTEIN   N00014-75-C-1111

UNCLASSIFIED   RR-162   NL

1 OF 1

AD
A069861

END
DATE
FILMED
7--79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

LEVEL II

(12)

LUX ET VERITAS

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

The Role of Object Primitives in Natural Language Processing

by

Wendy G. Lehnert and Mark H. Burstein

Research Report #162

May 1979

The research described here was doen at the Yale Artificial Intelligence Project and is funded in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 14 RR-162	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): 6 The Role of Object Primitives in Natural Language Processing.		5. TYPE OF REPORT & PERIOD COVERED: 9 Technical Report
		6. PERFORMING ORG. REPORT NUMBER: 15
7. AUTHOR(s): 10 Wendy G. Lehnert and Mark H. Burstein		8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-1111
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Yale University - Department of Computer Science, 10 Hillhouse Avenue, New Haven, Connecticut 06520		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 11 May 79
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, Virginia 22209		12. REPORT DATE: January 79
		13. NUMBER OF PAGES: 26
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems Program, Arlington, Virginia 22217 — 12 33p.		15. SECURITY CLASS. (of this report): Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Associative Memory
Memory Organization
Knowledge Representation
Inference Generation
Conceptual Analysis
Cognition
Natural Language Understanding

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Natural language processing techniques rely in part on the use of functional knowledge about physical objects, and an associative memory structure. In this paper, a representational system called Object Primitives is presented as an extension to the system of Conceptual Dependency for the purpose of representing physical objects and providing an organizing structure for associative memory. A computer program, OPUS, is described which applies this representational system to the problem of analyzing natural language

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407 051

sentences dealing with objects. Inferences derived from Object Primitive descriptions are made during the conceptual analysis by a system of demons, providing a framework for an integrated understanding system.

-- OFFICIAL DISTIRUBTION LIST --

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 copies
Advanced Research Projects Agency Cybernetics Technology Office 1400 Wilson Boulevard Arlington, Virginia 22209	3 copies
Office of Naval Research Branch Office - Boston 495 Summer Street Boston, Massachusetts 02210	1 copy
Office of Naval Research Branch Office - Chicago 536 South Clark Street Chicago, Illinois 60615	1 copy
Office of Naval Research Branch Office - Pasadena 1030 East Green Street Pasadena, California 91106	1 copy
Mr. Steven Wong Administrative Contracting Officer New York Area Office 715 Broadway - 5th Floor New York, New York 10003	1 copy
Naval Research Laboratory Technical Information Division Code 2627 Washington, D.C. 20375	6 copies
Dr. A.L. Slafkosky Scientific Advisor Commandant of the Marine Corps Code RD-1 Washington, D.C. 20380	1 copy
Office of Naval Research Code 455 Arlington, Virginia 22217	1 copy

Office of Naval Research — 1 copy
Code 458
Arlington, Virginia 22217

Naval Electronics Laboratory Center — 1 copy
Advanced Software Technology Division
Code 5200
San Diego, California 92152

Mr. E.H. Gleissner — 1 copy
Naval Ship Research and Development
Computation and Mathematics Department
Bethesda, Maryland 20084

Captain Grace M. Hopper — 1 copy
NAICOM/MIS Planning Board
Office of the Chief of Naval Operations
Washington, D.C. 20350

Mr. Kin B. Thompson — 1 copy
Technical Director
Information Systems Division
OP-91T
Office of the Chief of Naval Operations
Washington, D.C. 20350

Advanced Research Project Agency — 1 copy
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209

Professor Omar Wing — 1 copy
Columbia University in the City of New York
Department of Electrical Engineering and
Computer Science
New York, New York 10027

Office of Naval Research — 1 copy
Assistant Chief for Technology
Code 200
Arlington, Virginia 22217

Captain Richard L. Martin, USN — 1 copy
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York 09501

Major J.P. Pennell — 1 copy
Headquarters, Marine Corp.
(Attn: Code CCA-40)
Washington, D.C. 20380

The Role of Object Primitives
in Natural Language Processing

Wendy G. Lehnert and Mark H. Burstein

Computer Science Department
Yale University
Box 2158 Yale Station
New Haven, CT 06520

January, 1979

ABSTRACT

Natural language processing techniques rely in part on the use of functional knowledge about physical objects, and an associative memory structure. In this paper, a representational system called Object Primitives is presented as an extension to the system of Conceptual Dependency for the purpose of representing physical objects and providing an organizing structure for associative memory. A computer program, OPUS, is described which applies this representational system to the problem of analyzing natural language sentences dealing with objects. Inferences derived from Object Primitive descriptions are made during the conceptual analysis by a system of demons, providing a framework for an integrated understanding system.

Topics and Key Words

Topics and key words relevant to this paper are associative memory, memory organization, knowledge representation, inference generation, conceptual analysis, cognition, and natural language understanding.

The Role of Object Primitives
in Natural Language Processing*

by

Wendy G. Lehnert

and

Mark H. Burstein

1. INTRODUCTION

It is widely recognized that the process of understanding natural language texts cannot be accomplished without accessing mundane knowledge about the world [Bobrow, et al. 1977, Charniak 1972, Norman, et al. 1975, Minsky, M. 1975]. That is, in order to resolve ambiguities, build expectations, and make causal connections between events, we must make use of all sorts of episodic, stereotypic and factual knowledge of our world. In this paper we will concentrate on knowledge about physical

----

*This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.

objects. In particular, we will see how functional knowledge about physical objects and associations between physical objects can be exploited in an understanding system.

Consider the sentence

(1) John opened the bottle and poured the wine.

Anyone reading and understanding this sentence makes assumptions about what happened which go far beyond what is actually stated. For example, we assume without hesitation that the wine being poured came from inside the bottle. Although this seems quite obvious, in fact there are several other interpretations which fit what is actually stated. First of all, there is no reason to assume that the wine was initially in the bottle. John could be filling the bottle from some other container rather than emptying it. Alternatively, the wine being poured could have nothing to do with the bottle that was opened. John could be opening one bottle and pouring wine from an entirely different bottle. There is nothing in a literal reading of the sentence to prevent either of these interpretations, but some cognitive inference mechanism forces us (as human understanders) to connect these two events in a causal construction.

In addition to the assumptions made about where the wine comes from, we rely on our knowledge of bottles and what it means for a bottle to be "open", when interpreting the sentence. Only by drawing on this knowledge of states are we able to conclude

that John had to open the bottle in order to pour the wine out of the bottle. Strong associations are at work helping us to make these connections. Even for the sentence

(2) John closed the bottle and poured the wine.

we are inclined to assume that the wine is in the bottle, before we realize that this leads to a contradiction. The fact that we recognize a contradiction here indicates that we are utilizing knowledge about closed bottles and what cannot be done with closed bottles. In fact, we will show how specific expectations derived from our knowledge of open bottles are responsible for the natural human interpretation of "John opened the bottle and poured the wine."

Now consider the sentence

(3) John turned on the faucet and filled his glass.

We know immediately on hearing this that John filled his glass with water from the faucet. Yet, not only is water never mentioned in the sentence, but there is nothing in the sentence which explicitly relates turning on the faucet and filling the glass. The glass could conceivably be filled with milk from a carton. However, in the absence of some greater context which forces a different interpretation on us, we immediately assume that the glass is being filled with water from the faucet.

The question, then, is what knowledge of bottles, wine and faucets enables us to interpret these sentences as we do? What cognitive processes at the time of understanding cause us to conclude without hesitation that the wine was poured from the bottle and the glass was filled with water from the faucet? This paper describes a computer program which processes sentences such as those above, to arrive at meaning representations which include those assumptions that a human understander would make. To do this requires the use of stereotypic knowledge of physical objects. This information is captured in OPUS (Object Primitive Understanding System) by using a set of conceptual primitives called Object Primitives [Lehnert 1978]. Object Primitives (OP) were designed to act in conjunction with Schank's conceptual dependency system of representation [Schank 1975]. The processes developed to perform conceptual analysis in OPUS involve the integration of a conceptual analyzer similar to Riesbeck's ELI [Riesbeck and Schank 1976] with demon-like procedures for memory interaction and the introduction of object-related expectations.

2. Object Primitives

The primary focus in this research has been on the development of processes which utilize information provided by Object Primitives to facilitate the "comprehension" of natural language texts by computer. That is, we were primarily concerned with the introduction of stereotypic knowledge of objects into the conceptual analysis of text. By using information stored in

OP descriptions, we were able to increase the interpretive power of the analyzer in order to handle sentences of the sort discussed earlier.

What follows is a brief description of the seven Object Primitives. A more thorough discussion can be found in [Lehnert 1978]. For those unfamiliar with the primitive acts of Schank's conceptual dependency representation, a discussion of those can be found in [Schank 1975].

The Object Primitive CONNECTOR is used to indicate what actions (described in terms of Schank's primitives acts) are normally enabled when an object is in a particular state. A CONNECTOR enables a transfer between two spatial regions. For example, a window and a door are both CONNECTORs which enable the PTRANSing (physical transfer) of objects through them when they are open. In addition, a window is a CONNECTOR which enables an ATTEND (of eyes) or an MTRANS (mental transfer) with instrument ATTEND (of eyes). These events are enabled regardless of whether the window is open or closed. That is, one can see through a window, and therefore read or observe things on the other side, even when the window is closed. In the examples discussed above, the open bottle is given a CONNECTOR description. This will be discussed further later.

A SEPARATOR disenables a transfer between two spatial regions. A closed door and a closed window are both SEPARATORs which disenable the act PTRANS from one region to another. In addition, a closed door is a SEPARATOR which disenables the acts

MTRANS with instruments ATTEND eyes (unless the door is transparent) or ears. That is, one is normally prevented from seeing or hearing through a closed door. Similarly, a closed window is a SEPARATOR which disenables MTRANS with instrument ATTEND ears, although, as mentioned above, one can still see through a closed window to the other side. A closed bottle is another example of an object with a SEPARATOR description.

It should be clear by now that objects described using Object Primitives are not generally described by a single primitive. In fact, not one but several sets of primitive descriptions may be required. This is illustrated above by the combination of CONNECTOR and SEPARATOR descriptions required for a closed window, while a somewhat different set are required for an open window. These sets of descriptions form a small set of "states" which the object may be in. This representational system effectively treats open and closed windows as conceptually distinct objects in spite of the fact that our lexical expressions suggest that we have one fixed object assuming two different states.

A SOURCE description indicates that a major function of the object described is to provide the user of that object with some other object. Thus a faucet is a SOURCE of water, a wine bottle is a SOURCE of wine, and a lamp is a SOURCE of the phenomenon called light. SOURCEs often require some sort of activation. Faucets must be turned on, wine bottles must be opened, and lamps are either turned on or lit depending on whether or not they are

electric.

The Object Primitive CONSUMER is used to describe objects whose primary function is to consume other objects. A trash can is a CONSUMER of waste paper, a drain is a CONSUMER of liquids, and a mailbox is a CONSUMER of mail. Some objects are both SOURCEs and CONSUMERs. A pipe is a CONSUMER of tobacco and a SOURCE of smoke. An ice cube tray is a CONSUMER of water and a SOURCE of ice cubes.

Many objects can be described in part by relationships that they assume with some other objects. These relations are described using the Object Primitive RELATIONAL. Containers, such as bottles, rooms, cars, etc., have as part of their descriptions a containment relation, which may specify defaults for the type of object contained. Objects, such as tables and chairs, which are commonly used to support other objects will be described with a support relation.

Objects such as buildings, cars, airplanes, stores, etc., are all things which can contain people. As such, they are often distinguished by the activities which people in those places engage in. One important way of encoding those activities is by referring to the scripts which describe them. The Object Primitive SETTING is used to capture the associations between a place and any script-like activities that normally occur there. It can also be used to indicate other, related SETTINGs which the object may be a part of. For example, a dining car will have both a restaurant script and an associated SETTING of passenger

train. This information is important for the establishment of possible contexts, and the many domain specific expectations which will therefore be available to guide processing at both the conceptual analysis or word definitional level and when making inferences at higher levels of cognitive processing.

The final Object Primitive, GESTALT, is used to characterize objects which have recognizable, and separable, subparts. Trains, hi-fi systems, and kitchens, all evoke images of objects characterizable by describing their subparts, and the way that those subparts relate to form the whole. The Object Primitive GESTALT is used to capture this type of description.

Using this set of primitives as the foundation for a memory representation, we can construct an associative memory by introducing associative links external to object primitive decompositions [Lehnert 1978]. We have already achieved a class of associations within object primitive decompositions. For example, the conceptual description of a wine bottle will include a SOURCE description for a bottle in which the SOURCE output is specified as wine. This amounts to an associative link from the concept of a wine bottle to the concept of wine. But how can we construct an associative link from wine back to wine bottles? Wine does not have an object primitive decomposition which involves wine bottles, so we must resort to some construction which is external to object primitive decompositions.

Four associative links have been proposed [Lehnert 1978], each of which points to a particular object primitive description. For the problem of wine and wine bottles, an associative OUTPUTFROM link is directed from wine to the SOURCE description of a wine bottle. This external link provides us with an associative link from wine to wine bottles.

3. The Program

I will now describe the processing of two sentences very similar to those discussed earlier. The computer program (OPUS) which performs the following analyses was developed using a conceptual analyzer written by Larry Birnbaum [Birnbaum and Selfridge 1978]. OPUS was then extended to include a capacity for setting up and firing "demons" or "triggers" as they are called in KRL [Bobrow and Winograd 1977]. The functioning of these demons will be illustrated below.

3.1 The Initial Analysis

We will first look at the processing for "John opened the bottle so he could pour the wine," in detail. The phrase "John opened the bottle," is analyzed to produce the following representation:

```
*John* <=> *DO*

             result
*bottle* <=> CONNECTOR
               ENABLES
                                           -> ?X
          ?HUMO <=> PTRANS <- ?OBJ <-
                                           -< (INSIDE PART SELF)
   (or)        ENABLES
                                           -> (INSIDE PART SELF)
          ?HUMO <=> PTRANS <- ?OBJ <-
                                           -< ?Y
   (or)        ENABLES
                                             -> ?OBJ is inside SELF
          ?HUMO <=> ATTEND <- ?SENSE <-
                                             -<
```

Where SELF refers to the object being described (the bottle) and ?--- indicates an unfilled slot.

John represents an internal memory representation for a person with the name John, and *bottle* points to a memory token constructed for the bottle mentioned. These memory tokens for John and the bottle are constructed by a general demon which is triggered during conceptual analysis whenever a PP (the internal representation for an object) is introduced.

The above diagram represents the assertion that John did something which caused the bottle to assume a state where its CONNECTOR description applies. The CONNECTOR description indicates that something can be removed from the bottle (PTRANS FROM (INSIDE PART SELF)), put into the bottle (PTRANS TO (INSIDE PART SELF)), or its contents can be smelled, looked at, or generally examined by some sense modality (ATTEND). This CONNECTOR description is not part of the definition of the word 'open'. It is specific knowledge that people have about what it means to say that a bottle is open. It is not even the case that

opening something always builds a CONNECTOR description. For example, an open umbrella is described using the Object Primitive SEPARATOR since an open umbrella disenables rain from falling on the person holding the umbrella.

In arriving at the above representation, the program must retrieve from memory the OP description of what it means for a bottle to be open. This information is stored beneath its prototype for bottles, *BOTTLE*. Presumably, there is also script-like information about the different methods for opening bottles, the different types of caps (corks, twist-off, ...), and which method is appropriate for which cap. However, for the purpose of understanding a text which does not refer to a specific type of bottle, cap, or opening procedure, what is important is the information about how the bottle can then be used once it is opened. This is the kind of knowledge that Object Primitives were designed to capture.

When the analyzer builds the state description of the bottle, a general demon associated with new state descriptions is triggered. This demon is responsible for updating memory by adding the new state information to the token in the ACTOR slot of the state description. Thus the bottle token is updated to include the given CONNECTOR description. For the purposes of this program, the bottle is then considered to be an "open" bottle. A second function of this demon is to set up expectations for future actions based on the new information. In this case, templates for three actions the program might expect

to see described can be constructed from the three partially specified conceptualizations shown above in the CONNECTOR description of the open bottle. These templates are attached to the state description as possible consequences of that state, for use when attempting to infer the causal connections between events.

```
*bottle* <=> CONNECTOR
          ^
          |   (possible enabled actions)
          |
[                                        -> ?X                       ]
[ ?HUMO <=> PTRANS <- ?OBJ <-|                                       ]
[                                        -< (INSIDE PART *bottle*)   ]
[                                        -> (INSIDE PART *bottle*)   ]
[ ?HUMO <=> PTRANS <- ?OBJ <-|                                       ]
[                                        -< ?Y                       ]
[                                          -> ?OBJ is inside *bottle*]
[ ?HUMO <=> ATTEND <- ?SENSE <-|                                     ]
[                                          -<                        ]
```

The remainder of the sentence reads "so he could pour the wine." The "so...could" indicates that there should be some direct or indirect enabling relationship between the action described in the first part of the sentence and the one currently being described. This is captured in the program by setting up a special procedure to verify a causal connection. This procedure will only be activated when the sentence is completed. It attempts to find the connection between the two conceptualizations pointed to by the ENABLE link formed when the analyzer saw the "so ... could"; the state of the bottle being open, and the act of pouring the wine.

3.2 Concept Driven Inferences

The phrase "so he could pour the wine." is analyzed as

```
           ⇑ enable          ┌> ?X
*John* <=> PTRANS <- *wine* <┤
                             └< (INSIDE PART ?CONTAINER)
```

When this representation is built by the analyzer, some slots remain as yet unfilled. For example, we do not know that the the wine being poured is coming from the previously mentioned bottle. This inference is made in the program by a slot-filling demon called the CONTAINER-FINDER, attached to the primitive act PTRANS. The demon is triggered when a PTRANS is built whose conceptual object is some substance being described as coming from inside an unspecified container. When this occurs, a request is generated which looks on the list of active tokens (a part of short term memory) for any containers that might contain the given substance. A suitable container is one in which we might normally expect to find the particular substance, in this case wine.

There are two ways to recognize a possible container: [1] via the DEFAULT-CONTAINMENT test, or [2] via the COMMON-SOURCE test. The DEFAULT-CONTAINMENT test relies on an associative link from the RELATIONAL description of a container to a default substance within that description. The COMMON-SOURCE test relies on an associative OUTPUTFROM link from the substance to the

SOURCE description of a container for that substance.

At different times, either the DEFAULT-CONTAINMENT test or the COMMON-SOURCE test may be necessary in order to establish probable containment. For example, it is reasonable to expect a vase to contain water since the RELATIONAL description of a vase has default containment slots for water and flowers. But we do not always expect water to come from vases since there is no OUTPUTFROM link from water to a SOURCE description of a vase. If we heard "Water spilled when John bumped the vase," containment would be established by the DEFAULT-CONTAINMENT test. Associative links are not always bi-directional (vase ---> water, but water -/-> vase) and we need separate mechanisms to trace links with different orientations. In our wine example, the COMMON-SOURCE test is responsible for establishing containment, since wine is known to be OUTPUTFROM bottles but bottles are not always assumed to hold wine.

Another expectation which is fulfilled during initial analysis is a request looking for the contents of the bottle mentioned in the first clause of the sentence. This expectation was set up by a demon called the CONTENTS-FINDER when the description of the open *bottle* was built, and is independent from the CONTAINER-FINDER demon activated by pouring. This demon is activated by a state description for an open container where the container is a common SOURCE object for which no known output exists. The presence of the SOURCE object can be explained if its output is subsequently used in the text. Here again, the

list of active tokens is searched for an object which could be OUTPUT-FROM a bottle, and the token for this particular bottle is then marked as being a SOURCE of that object. The description of this particular bottle as a SOURCE of wine is equivalent, in Object Primitive terms, to saying that the bottle is a wine bottle.

3.3 Causal Verification

Once the requests trying to fill slots not filled during the initial analysis have been considered, the process set up by "so...could" which attempts to make causal connections between conceptualizations is activated.

```
*bottle* <=> CONNECTOR
           ^
           |  enable?...(demon) ====> [activate verifier]
           |
           |                         --> ?X
*John*  <=> PTRANS <- *wine* <--|
                                     --< (INSIDE PART *bottle*)
```

This process first looks for a match between the conceptual representation for the enabled action (pouring the wine), and one of the potentially enabled acts under the state description resulting from John opening the bottle. In this case, the potentially enabled acts result from the description of the open bottle as a CONNECTOR. In other cases, the Object Primitives SEPARATOR, SOURCE, and CONSUMER may be responsible for similar kinds of expectations. In this example, a match is immediately found between the action of pouring from the bottle (the PTRANS

built by the analyzer, with FROM slot filled in CONTAINER-FINDER demon) and the expected action generated from the CONNECTOR description of an open bottle (PTRANS FROM (INSIDE PART SELF)).

When a match is found, further conceptual checks are made on the enabled act to ensure that the objects found in the slots of that act fit criteria not relating to their positions in the conceptual dependency representation. When the match is based on expectations derived from the CONNECTOR description of a container, the check is a "container/contents check," which attempts to ensure that the object found in the container may reasonably be expected to be found there. The sentence "John opened the bottle so he could pull out the elephant", is peculiar because we have no expectations that elephants are found in bottles, even if the bottle was big enough. The strangeness of this sentence can only be explained by the application of stereotypic knowledge about what we expect and don't expect to find inside a bottle.

The container/contents check is very similar to the test described above in connection with the CONTAINER-FINDER demon which was used to find the bottle John poured from. That is, the bottle is checked by both the DEFAULT-CONTAINMENT test and the COMMON-SOURCE test for any known default links relating wine and bottles. When this check succeeds, the the enable link has been verified by matching an expected action, and by checking restrictions on related objects appearing in the slots of that action. The two CD acts that matched are then merged.

The merging process accomplishes several things. First, it completes the linking of the causal chain between the events described in the sentence. Secondly, it can cause the filling of empty slots appearing in either the enabled act or in the enabling act, wherever one left a slot unspecified, and the other had that slot filled. An example of this is found in the next section.

3.4 Causal Chain Construction

In the last example, a causal connection was made as a result of a direct match between one of the expected actions resulting from the opening of the bottle, and what actually occurred. However, making causal connections between events is seldom that simple. In our next example, further inferences must be made to complete the causal chain between the conceptualizations derived from the input.

The sentence

(4) John turned on the faucet so he could drink.

has several interesting differences from the one whose processing was described above. First, turning on a faucet does not build a CONNECTOR description at all. Rather, a faucet is described as a SOURCE of water. In general, things which must be "turned on" to be used have a SOURCE description as part of their OP

decomposition.

The representation produced by the conceptual analyzer for "John turned on the faucet," is

```
      *John* <=> *DO*
                 ⇑
                 ‖ result
    *faucet* <=> (SOURCE with OUTPUT = *water*)
```

As with the bottle in the previous example, the description of the faucet as an active SOURCE of water is based on information found beneath the prototype for faucet, describing the "on" state for that object.

The demon triggered by new state descriptions is activated again here, this time setting up expectations based on the new SOURCE description of the object, rather than a CONNECTOR description. The principle expectation for SOURCE objects is that the person who "turned on" the SOURCE object wants to take control of (and ultimately make use of) whatever it is that is output from that SOURCE. In CD, this is expressed by a template for an ATRANS (abstract transfer) of the output object, in this case, water.

```
*faucet* <=> (SOURCE with OUTPUT = *water*)
           ⇑
           ‖ (possible enabled action)
           ‖
           ‖                           ┌-> ?HUMO
  ?HUMO <=> ATRANS <- *water* <-┤
                                       └-<
```

An important side effect of the construction of this expectation is that a token for some water is created. As we shall soon see,

the demon that fills in the object slot of the INGEST act depends on the presence of a token for water in STM to be able to fill that slot.

The representation for "he could drink" is simply

```
                                             ┌-> (INSIDE PART *John*)
*John* <=> INGEST <- ?LIQUID <-┤
             ^                               └-< (*mouth* PART *John*)
             |
             | inst                          ┌-> (*mouth* PART *John*)
*John* <=> PTRANS <- ?LIQUID <-┤
                                             └-< ?Y
```

Here again, there are some slots left unfilled when the conceptual analysis is completed. A special request to look for the missing liquid is set up by a demon on the act INGEST, similar to the one on the PTRANS in the previous example. This request finds the token for water placed in the short term memory when the expectation that someone would take control of some water was generated. The ability to make this inference is, therefore, indirectly dependent on the active, forward expectation that water will be used when someone turns on a faucet.

The causal chain completion that occurs for this sentence is somewhat more complicated than it was for the previous case. As we have seen, the only expectation set up by the SOURCE description of the faucet was for an ATRANS of water from the faucet. However, the action that is described here is an INGEST with instrumental PTRANS. When the chain connector fails to find a match between the ATRANS and either the INGEST or the

instrumental PTRANS, inference procedures are called in order to generate any obvious intermediate states that might connect these two acts.

```
         *faucet* <=> SOURCE with OUTPUT *water*
                   ^
                   ||
                   ||
                   ||  (possible enabled act)
    . . . . ?HUMO <=> ATRANS <- *water* <-------+
    .              ^                            |
    .              ||                           |
forward            ||                           |
inference          ||  result                   |
    .              ||                      (no match)
    .              ||                           |
    . . .> *water* <=> (POSS-BY ?HUMO)          |
                   ^                            |
                    \                           v
                     <------- match? --> *John* <=> INGEST <- ?LIQUID
                                                 ^
                                                 |  inst
                                         *John* <=> PTRANS <- ?LIQUID
```

The first inference rule that is applied is the resultative inference [Rieger 1975] that an ATRANS of an object TO someone results in a state where the object is possessed by (POSS-BY) that person. Once this state has been generated, it is matched against the INGEST in the same way the ATRANS was. When this match fails, no further forward inferences are generated, since simple possession of water can lead to a wide range of new actions, but no single specific one is strongly expected.

The backward chaining inferencer is then called to generate any known preconditions for the INGEST which was the final act mentioned in the sentence. The primary precondition (by causative inference) for being able to drink is, of course, that the person doing the drinking has the liquid which he or she is about to drink. Thus, a CD representation of the possession of

the water by John is inferred as an enabling state for his drinking, and this state is found to match the state (someone possesses water) inferred going forward from the ATRANS. This match causes the causal chain to be completed, and permits further slot specification in our memory representation. Due to the match of (someone) against (*John*), the program deduces that it was probably John who took (ATRANSed) the water from the faucet, in addition to turning it on. If the sentence had been "John turned on the faucet so Mary could drink.", then the assumption would have been that Mary took the water, not John.

```
          *faucet* <=> (SOURCE with OUTPUT = *water*)

                       enable                       -> ?HUMO
             ?HUMO <=> ATRANS <- *water* <-|
                                            -<

                       result
           *water* <=> (POSS-BY ?HUMO)
                                             match?
                                               yes...infer ?HUMO = *John*
      ...>*water* <=> (POSS-BY *John*)

backward
inference              enable
                                                 -> (INSIDE PART *John*)
      ..... *John* <=> INGEST <- ?LIQUID <-|
                                                 -< (*mouth* PART *John*)

                      inst                       -> (*mouth* PART *John*)
            *John* <=> PTRANS <- ?LIQUID <-|
                                                 -< ?Y
```

One should note here that the additional inferences used to complete the causal chain were very basic. The primary connections came directly from object-specific expectations derived from the Object Primitive descriptions of the objects involved.

IV. CONCLUSIONS

It is important to understand how OPUS differs from previous inference strategies in natural language processing. To emphasize the original contributions of OPUS we will compare it to Rieger's early work on inference and causal chain construction. Since Rieger's research is closely related to OPUS, a comparison of this system to Rieger's program will illustrate which aspects of OPUS are novel, and which aspects have been inherited.

There is a great deal of similarity between the types of inferences used in OPUS and those used by Rieger in his description of MEMORY [Rieger 1975]. The causative and resultative inferences used to complete the causal chain in our last example came directly from that work. In addition, the demons used by OPUS are similar in flavor to the forward inferences and specification (slot-filling) inferences described by Rieger. Expectations are explicitly represented here as they were there, allowing them to be used in more than one way, as in the case where water is inferred to be the INGESTed liquid solely from its presence in a previous expectation.

There are, however, two ways in which OPUS departs from the inference strategies of MEMORY in significant ways. [1] On one the level of computer implementation there is a reorganization of process control in OPUS, and [2] on a theoretical level OPUS exploits an additional representational system which allows

inference generation to be more strongly directed and controlled.

In terms of implementation, OPUS integrates the processes of conceptual analysis and memory-based inference processing. By using demons, inferences can be made during conceptual analysis, as the conceptual memory representations are generated. This eliminates much of the need for an inference discrimination procedure acting on completely pre-analyzed conceptualizations produced by a separate program module. In MEMORY, the processes of conceptual analysis and inference generation were sharply modularized for reasons which were more pragmatic than theoretical. Enough is known about the interactions of analysis and inference at this time for us to approach the two as concurrent processes which share control and contribute to each other in a very dynamic manner. Ideas from KRL [Bobrow and Winograd 1977] were instrumental in designing an integration of previously separate processing modules.

On a more theoretical level, the inference processes used for causal chain completion in OPUS are more highly constrained than was possible in Rieger's system. In MEMORY, all possible inferences were made for each new conceptualization which was input to the program. Initially, input consisted of concepts coming from the parser. MEMORY then attempted to make inferences from the conceptualizations which it itself had produced, repeating this cycle until no new inferences could be generated. Causal chains were connected when matches were found between inferred concepts and concepts already stored in its memory.

However, the inference mechanisms used were in no way directed specifically to the task of making connections between concepts found in its input text. This lead to a combinatorial explosion in the number of inferences made from each new input.

In OPUS, forward expectations are based on specific associations from the objects mentioned, and only when the objects in the text are described in a manner that indicates they are being used functionally. In addition, no more than one or two levels of forward or backward inferences are made before the procedure is exhausted. The system stops once a match is made or it runs out of highly probable inferences to make. Thus, there is no chance for the kinds of combinatorial explosion Rieger experienced. By strengthening the representation, and exploiting an integrated processing strategy, the combinatorial explosion problem can be eliminated.

OPUS makes use of a well structured set of memory associations for objects, the Object Primitives, to encode information which can be used in a variety of Rieger's general inference classes. Because this information is directly associated with memory representations for the objects, rather than being embodied in disconnected inference rules elsewhere, appropriate inferences for the objects mentioned can be found directly. By using this extended representational system, we can begin to examine the kinds of associative memory required to produce what appeared from Rieger's model to be the "tremendous amount of 'hidden' computation" necessary for the processing of

any natural language text.

REFERENCES

Birnbaum, L., and Selfridge M. (1978). On Conceptual Analysis. (unpublished) Yale University, New Haven, CT.

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, a frame driven dialog system. Artificial Intelligence, Vol. 8, No. 1.

Bobrow, D. G., and Winograd, T. (1977). An overview of KRL, a knowledge representation language. Cognitive Science 1, no. 1

Charniak, E. (1972). Toward a model of childrens story comprehension. AITR-266, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

Lehnert, W. G. (1978). Representing physical objects in memory. Technical Report #131. Dept. of Computer Science, Yale University, New Haven, CT.

Minsky, M. (1975). A framework for representing knowledge. In Winston, P. H., ed., The Psychology of Computer Vision, McGraw-Hill, New York, NY.

Norman, D. A., and Rumelhart, D. E., and the LNR Research Group (1975) Explorations in Cognition. W. H. Freeman and Co., San Fransisco.

Rieger, C. (1974). Conceptual memory. Ph.D. Thesis, Computer Science Department, Stanford University, Stanford CA.

Rieger, C. (1975). Conceptual memory. In R. C. Schank, ed., Conceptual Information Processing. North Holland, Amsterdam.

Riesbeck, C. and Schank, R. C. (1976). Comprehension by computer: expectation-based analysis of sentences in context. Technical Report #78. Dept. of Computer Science, Yale University, New Haven, CT.

Schank, R. C. (1975). Conceptual Information Processing. North Holland, Amsterdam.

Schank, R. C. and Abelson, R. P. (1977). Scripts, Plans, Goals, and Understanding. Lawrence Erlbaum Press, Hillsdale, NJ.

Schank, R. C., (1973). Identification of conceptualizations underlying natural language. In Schank, R. C. and Colby, K., eds., Computer Models of Thought and Language. W. H. Freeman and Co., San Fransisco.